



PROGRAMMING Expert

If you want to manipulate information within images you need an image-processing program. Mike James shows you how the Windows Image Acquisition Library can help you to put data from an image on to a PC

PROGRAMMING NEWS

EXTRA FEATURES FOR VISUAL STUDIO 2005

The new version of Microsoft's main development environment will have more features than originally advertised when it finally appears. Earlier this year, Microsoft pushed back the release date to the second half of 2005, making the project more than a year late. But according to corporate vice-president Soma Somasegar, various issues raised by testers have been fixed.

The most important new feature is 'edit and continue', which allows programmers to debug faster by stopping a running program, making a change and then continuing the code from where it left off. Web Form templates have been updated to avoid obsolete HTML and improve behaviour. Visual Studio supports all .NET languages including VB .NET and C#.

For more information, visit <http://blogs.msdn.com/somasegar> and <http://lab.msdn.microsoft.com/vs2005>.

DESPERATELY SEEKING SOFTWARE

Programmers seeking example code can draw on a huge resource of open-source software using the new Koders search engine. Koders indexes just under 200 million lines of public-source code in a wide range of languages.

Open-source software is an excellent resource; programmers can use someone else's source code and modify it as required. Programmers can also use it to demonstrate how to use an API or hardware that documentation doesn't describe adequately.

Koders' searches can be restricted to specific languages and licence types. It searches code repositories in universities and standard groups such as Apache, Mozilla, Novell Forge and SourceForge. As well as searching project descriptions it also searches the source code. This means you can type in the name of an API function, class library or DLL and find code that refers to it. The service is currently free, but there are plans for an enhanced enterprise edition that will probably be charged for. Visit www.koders.com for further details.

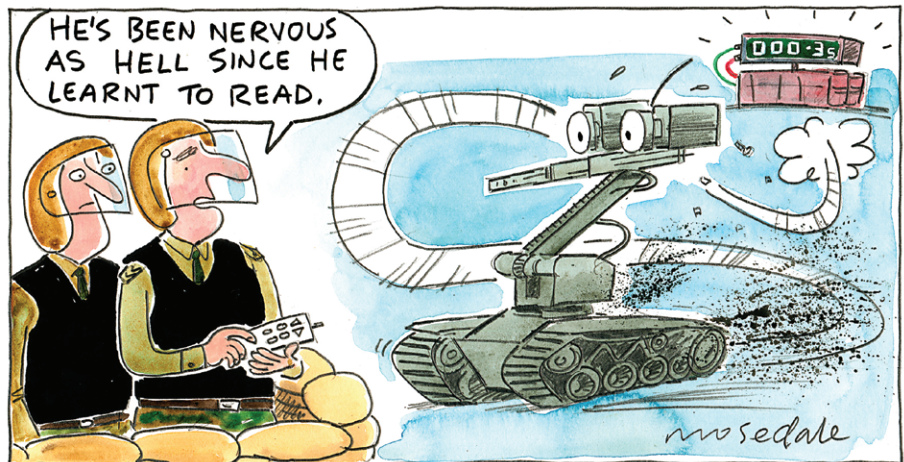


Image-processing software that takes a picture from a camera and extracts information from it sounds very clever and you'd expect applications that perform face and character recognition to be extremely complicated. However, we are going to devise a basic image-processing program using just a webcam and VBScript.

We dreamt up this project when faced with the problem of getting data from a meter on to a computer. The meter couldn't be connected via a parallel or serial port or even a custom USB port. So we decided simply to point a camera at the meter and write a program to read the digits it was displaying. A suitable USB webcam can be bought for less than £10.

This may still seem a tough task. But while it involves a level of artificial intelligence – character recognition to be precise – you need only a little ingenuity. This program is a work in progress, but the techniques used are easy to understand and you should be able to adapt and improve them.

There are two problems to solve. The first is rather generic: how do you read in a still image from a video camera so you can work with it? The second is more specific: how do you recognise digits in an image? To tackle the first problem, we used the Windows Image Acquisition (WIA) library. This works with any COM/ActiveX language and we've used VBScript in an HTA application.

DIGIT VISION

The WIA application program interface (API) was introduced in Windows Me and developed in Windows XP. It is responsible for many of the new video-based multimedia features available in both operating systems. Many features of the API used in our example may work under Me, but the documentation states that XP is the minimum requirement. It's also claimed that the API needs at least Windows XP with Service Pack 1 installed to

work. Although the API is installed as standard, it's best to install the Automation 2.0 update. First, navigate to the Microsoft download site at www.microsoft.com/downloads and search for WIA Automation. This is only 500KB, but it has to be installed manually. Unzip the download into a suitable directory. Then copy the file `wiaaut.dll` to your System32 directory. This is usually found in `C:\Windows\System32`. From the command prompt in the System32 directory, run the command:

```
RegSvr32 WIAAut.dll
```

You'll also find the `wiaaut.chm` and `wiaaut.chi` help files, which you can copy to `\Windows\help`. Or you can just click on the .chm file to open it.

GETTING STARTED

As you may recall from previous issues of *Computer Shopper*, an HTA program is a mixture of HTML, which defines the user interface, and script. You can create HTA using any text editor such as Notepad or Word. To get started, enter the following HTML template and save the result as `MeterReader.HTA`:

```
<HTML>
<Head>
<Title>Meter Reader</Title>
<HTA:Application Id="HTA"
Applicationname="MeterReader"
Border="Thick
Borderstyle="Normal"
Caption="Yes"
Contextmenu="No"
Icon=""
Maximizebutton="Yes"
Minimizebutton="Yes"
Scroll="No"
Showintaskbar="Yes"
Singleinstance="No"
```



```

Sysmenu="Yes"
Version="1.0"
Windowstate="Normal"
</HTA:Application>
</Head>
<Body>
  <Job>
    <Script Language=VBScript>

```

The HTML that defines the user interface will go between the <Body> and <Job> tag. The script goes between the <Script> and </Script> tags. Run this HTA program by double-clicking it, and it will open and show you a blank page. Next we add some code.

First, we size the window when the form first loads. To do this, we need to add an event handler to the <Body> tag:

```
<Body onLoad="Form1_OnLoad()">
```

The OnLoad event handler should be entered between the script tags:

```

Sub Form1_OnLoad()
  window.MoveTo 100,200
  window.ResizeTo 800,400
End sub

```

Next we must make a connection to the video device. We need to create an instance of the WIA CommonDialog object and then allow the user to select the device to use:

```

Set CommonDialog=CreateObject( _
  "WIA.CommonDialog")
Dim dev
Set dev=CommonDialog.ShowSelectDevice

```

This raises the question of which devices are likely to be listed as usable. If you have a video device or scanner with a true Windows 2000/XP driver, it should be visible in the list and usable via the WIA. However, many video devices, especially older digital still cameras, use Windows 98 drivers that are just relabelled as Windows 2000/XP drivers.

LIVE AID

To show live video, we must add the VideoPreview control to the form. To do this, enter the following just after the <Body> tag:

```

<OBJECT id="VideoWindow1"
name="VideoWindow1"
height="288" width="352"
classid="clsid:0B5F2CC8-5E1E-44F9-
899B-3B789705AFCA">
</OBJECT>

```

The size, height and width is set to display the full video image that the camera captures. You will have to change this to suit the camera you are using. Now add one line of script to connect the video camera to the VideoPreview control so it displays what the camera sees:

```
VideoWindow1.Device=dev
```

If you run the program now, you will find you have live video in a window. If you don't want to give the user the chance to select the camera, you could connect to it directly using:

```

Set dev=DeviceManager.DeviceInfos.
Item(2).Connect

```

There are other dialog boxes that can be used to configure the camera. Check the WIA documentation for more details.

PICTURE PERFECT

Next we need to take a snapshot and convert it into a form with which we can work. We also need to give the user some feedback. We'll create an HTML image control and two buttons. Add the following to the user interface after the </Object> tag:

```

<IMG id="Image1" height="288"
width="352" alt="" src="">
<br>
<INPUT id="Button1" type="button"
value="Setup" name="Button1"
onClick="Command1_OnClick()">
<INPUT id="Button2" type="button"
value="Read" name="Button2"
onClick="Command2_OnClick()">

```

This HTML adds an Image control and two buttons. The first button is used to set up the image ready for reading and the second button initiates a reading. The Setup button simply transfers a snapshot to the image box with some added information that shows how everything is aligned. Let's deal with taking and transferring the snapshot next.

Taking the snapshot is easy. All you have to do is use the device's ExecuteCommand method with the appropriate code for the TakePicture command. The only problem is that we can't use any of the predefined constants to do this because they aren't available in an HTA program. The only way of finding out what the codes are is by reading the type library information manually:

```

Sub Command1_OnClick()
  Dim item
  Set item = dev.ExecuteCommand( _
    "{AF933CAC-ACAD-11D2-A093-
00C04F72DC3C}")

```

Now we have taken the picture, we have to transfer it to an ImageFile object:

```

Dim ImageFile
Set ImageFile= item.Transfer( _
  "{B96B3CAB-0728-11D3-9D7B-
0000F81EF32E}")

```

An ImageFile object is nothing more than raw image data as a file. To get useful data we need to convert it into a Vector object:

```

Dim vec
Set vec=ImageFile.ARGBData

```

The ARGBData property returns a vector. This is an array of long variables, each four bytes. Each element represents a pixel in alpha, red, green and blue format, one byte to each property. Alpha gives

the transparency of the image, so we can ignore it in this application and simply extract the RGB values. You can get at a single pixel by treating the vector as an array. For example, vec(33) returns the value of the 33rd pixel. To access a pixel with x,y coordinates, we will have to calculate where it is stored in the one-dimensional array. A function that does this is given later.

Now that we have a vector of pixel values, we can work with the image using a subroutine called SampleDigit. This routine is passed through the image – plus a few parameters to specify display size – and returns the recognised digit. You should now enter the following code:

```

h=ImageFile.Height
w=ImageFile.Width
x1=100
y1=100
w1=24
h1=70
s=4
Call SampleDigit(vec,x1,y1,h1,w1,s,w)

```

For reasons that will soon become clear, SampleDigit modifies the image stored in vec. The next line converts the modified image back to an ImageFile:

```
Set ImageFile=vec.ImageFile(w,h)
```

Finally we save the ImageFile on disk, but only after checking if the file exists and deleting it as necessary:

```

Set fso = CreateObject( _
  "Scripting.FileSystemObject")
If (fso.FileExists("Mask.BMP")) Then
  Set f2 = fso.GetFile("Mask.BMP")
  f2.Delete
End if
ImageFile.SaveFile "Mask.BMP"

```

Now the image is saved on disk, we can load it into the Image control:

```

Image1.Src="Mask.BMP"
End Sub

```

If you add a dummy SampleDigit routine, you can try out the program:

```

Function SampleDigit(v,left,top,height
,s,width,w)
End Function

```

If you click the Setup button, the live image is transferred to the Image control.

So now we know how to get the image we need. You can use these techniques within your own programs to manipulate or extract data from image files. The image can be obtained from a scanner with the same techniques.

READ IT YOURSELF

Now we come to the second part of our problem: reading the digits. For simplicity's sake, we're going to show you how to read just one digit of the display. Just repeat the process if you have more digits to read. You can read a single digit on a liquid crystal display (LCD) without resorting to complex pattern-recognition methods. Each digit is made



from a fixed seven-segment pattern of bars. To determine which digit is being displayed, we just have to determine whether each of the seven bars is on or off. Don't underestimate this task, though, as the image quality can be terrible. It's difficult to ensure that the display is evenly lit as even when the image looks clear enough, it can be difficult to read when captured with a video camera.

The simplest way of deciding if a bar is on or off is to sample a point inside the bar and a point outside the bar and compare the two. The two points must be close together as the lighting may vary if they are far apart and make them appear different even if they are the same. The names given to each of the bars can be seen in Figure 1 on the right, along with white lines that indicate where the measurements are made.

When pressing the Setup button, similar white lines should be drawn on the onscreen image to help the user align the camera and display. In a more sophisticated version of the program, it's better to allow the user to drag the crosswires on to the digit elements of the image. In our simple version, we'll leave it up to the user to align the image with the fixed crosswires by adjusting the camera position.

SAMPLE PLEASURES

Next we have to code the SampleDigit routine that turns an image into a single-digit number. Most of the work is done by two routines: GetHBar and GetVBar detect whether a horizontal bar or vertical bar in the display is illuminated. Most of the parameters of the SampleDigit routine are shown in Figure 2 on the right, where s is the thickness of each bar and w is the width of the entire image, stored in v. These parameters are set in the above code by the lines before SampleDigit is called.

There are two ways to make sure that they are correct for the picture you are recording. Either change the parameter values in the code or move the camera in and out until the crosswires fit the read-out image.

The SampleDigit routine is as follows. The variable t sets the threshold, which is the smallest brightness difference needed for a bar to be 'on':

```
Function SampleDigit(v,left,top,height
,width,s,w)
t=20
x=Int(left+width/2)
y=Int(top)
b1=GetHBar(v,x,y,s,t,w)
y=Int(top+height/2)
b2=GetHBar(v,x,y,s,t,w)
y=Int(top+height)
b3=GetHBar(v,x,y,-s,t,w)
x=Int(left)
y=Int(top+height/4)
b4=GetVBar(v,x,y,s,t,w)
y=Int(top+3*height/4)
b5=GetVBar(v,x,y,s,t,w)
x=Int(left+width)
y=Int(top+height/4)
b6=GetVBar(v,x,y,-s,t,w)
y=Int(top+3*height/4)
```

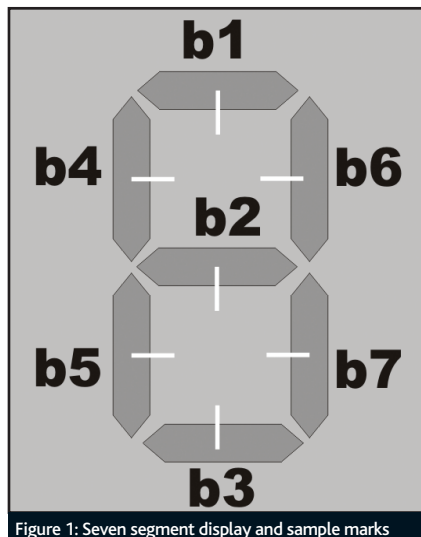


Figure 1: Seven segment display and sample marks

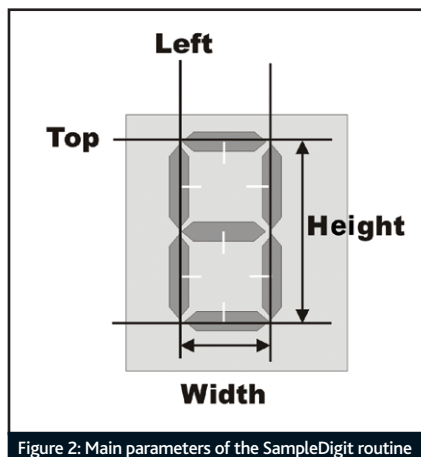


Figure 2: Main parameters of the SampleDigit routine

```
b7=GetVBar(v,x,y,-s,t,w)
End Function
```

Each segment of the code sets x and y to the coordinates of a bar, then uses GetHBar or GetVBar to check its state. GetHBar uses additional functions; GetGreyPixel retrieves the brightness of a pixel at x,y and PutVMark draws a white vertical dash on the image.

FADE TO GREY

By comparing the difference between GetGreyPixel inside the bar and outside the bar, GetHBar returns True if the difference is greater than t (the bar is lit) or False if the difference is less than t (the bar is not lit):

```
Function GetHBar(v,x,y,s,t,w)
p1=GetGreyPixel(v,x,y,w)
p2=GetGreyPixel(v,x,y+s,w)
Call PutVMark(v,x,y,s,w)
GetHBar=((p2-p1)>t)
End Function
```

GetVBar is similar but uses PutHMark to draw a horizontal dash:

```
Function GetVBar(v,x,y,s,t,w)
p1=GetGreyPixel(v,x,y,w)
p2=GetGreyPixel(v,x+s,y,w)
Call PutHMark(v,x,y,s,w)
GetVBar=((p2-p1)>t)
End Function
```

The GetGreyPixel function extracts the bytes corresponding to R, G and B using the AND operator, and combines them to give the total brightness. The $(x+(y-1)*w)$ expression gives the position in the vector array of the pixel with coordinates x,y. You can modify this function if you want a colour value for the pixel:

```
Function GetGreyPixel(v,x,y,w)
pixel=v(x+(y-1)*w)
R=(pixel AND &HFF0000)/CInt(65536)
G=(pixel AND &HFF00)/CInt(256)
B=pixel AND &HFF
GetGreyPixel=SQR(R*R+G*G+B*B)
End Function
```

PutPixel is used by the dash-drawing routines to mark a dot on the image. It is like GetGreyPixel in reverse:

```
Sub PutPixel(v,p,x,y,w)
v(x+(y-1)*w)=p
End sub
```

The PutVMark and PutHMark subroutines create a dash by calling PutPixel in a loop. The Sgn function allows the mark to be drawn in either direction from the starting position. A positive s draws in a positive direction and a negative s draws in the opposite direction:

```
Sub PutVMark(v,x,y,s,w)
For i=y To y+s Step Sgn(s)
CALL PutPixel(v,RGB(255,255,255),x,i,w)
Next i
End Sub

Sub PutHMark(v,x,y,s,w)
For i=x To x+s Step Sgn(s)
CALL PutPixel(v,RGB(255,255,255),i,y,w)
Next i
End Sub
```

If you run the program and click the Setup button, you will see the transferred image has a set of crosswires drawn on it. These allow the user to adjust the webcam correctly, so the crosswires are positioned over the digit as shown in Figure 3 opposite. Because the LCD is slightly tilted, the meter has to be tilted in the picture.

BAR DECODING

When you press the Setup button, the code takes measurements on all the bars but doesn't use this information because its main purpose is to draw crosswires. Once the crosswires have been aligned, we can read the display. To do this, we have to write the click event handler for the second button. This is similar to the Setup button but it doesn't transfer the marked-up image to the display:

```
Sub Command2_onClick()
Dim item
Set item = dev.ExecuteCommand( _
"[AF933CAC-ACAD-11D2-A093-00C04F72DC3C]")
Dim ImageFile
Set ImageFile= item.Transfer( _
"[B96B3CAB-0728-11D3-9D7B-0000F81EF32E]")
```




REVIEW BOOK

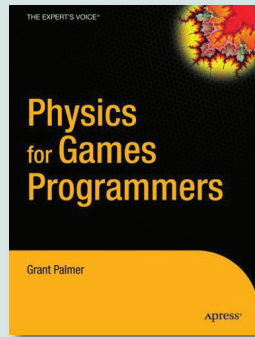
Physics for Games Programmers

RATING ★★★★★
PRICE £31
ISBN 159059472X
PAGES 350

Physics for Games Programmers is a decent read but it tries to be too simple, and the attempts at persuading you that the subject is fun seem forced. It starts by implying that maths isn't important, but then it moves on to calculus and differential equations.

Many of the ideas presented are illustrated by programs in Java. The book has a sticker on the front saying that the programs are in C# and C as well, but this is true only if you download them from the website.

The advanced physics is used well to explain how collisions work, including friction, how boats float and move, how fluids behave and how explosions work. If this subject matter



appeals, you will enjoy the book. However, it is debatable whether this level of detail is useful when creating games. Often it's too realistic, and simpler approximations work faster.

You probably don't need to know this much about physics to create a game, and you certainly don't need to know any of the general information that's included.

- ✓ Excellent overview of physics principles and simulation
- ✗ Goes too far for most game programming

REVIEW BOOK

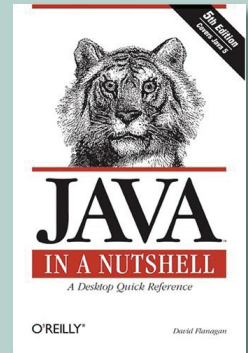
Java in a Nutshell

RATING ★★★★★
PRICE £31
ISBN 0596007736
PAGES 1,264

In this, the fifth edition of David Flanagan's *Java in a Nutshell*, the book has lost its original emphasis of trying to convert programmers from C or C++ to Java and concentrates on discussing the Java framework, tools and APIs. It now also covers Java 5.0.

Java 5.0 has two chapters to itself. The book's first chapters are mainly explanations, but the bulk is a rigidly formatted reference work.

There are a couple of problems with this book. The most evident is its size. This 'in a nutshell' reference is 1,264 pages long, and to squeeze all the information into its small page size it uses a tiny typeface and a cramped layout. We don't know what sort of nutshell the author has in mind, but it's not one we've ever seen.



The second problem is that the bulk of the book is a revised version of standard online documentation. There's no justification for creating paper reference works of this type.

We wonder if there will be a sixth edition. Still, it's recommended if you like paper reference works, or you are prepared to pay for the insights in the first few chapters.

- ✓ Well-written, comprehensive Java reference
- ✗ Huge for a book that claims to be 'in a nutshell'

```
Dim vec#
Set vec=ImageFile.ARGBData#
Dim h#
Dim w#
h=ImageFile.Height#
w=ImageFile.Width#
x1=100#
y1=100#
w1=24#
h1=65#
s=10#
Msgbox SampleDigit(vec,x1,y1,h1,w1
,s,w)#
End Sub#
```

The above routine assumes that the SampleDigit function returns the digit value and simply displays it in a message box. For SampleDigit to do this, it must decode the information stored in the variable b1 to b7 by the GetVBar and GetHBar calls.

There are several ways of doing this. You could use a lookup table for which bars are involved in displaying each digit, or you could use a binary tree decision procedure. This sounds complicated, but it's actually easy and well worth knowing about. In our example, this is how it works. To create a binary decision tree, you must find tests that are true for around half the cases and false for the others. We're looking for bars that are lit for about half of the possible digits. Each such test eliminates half the

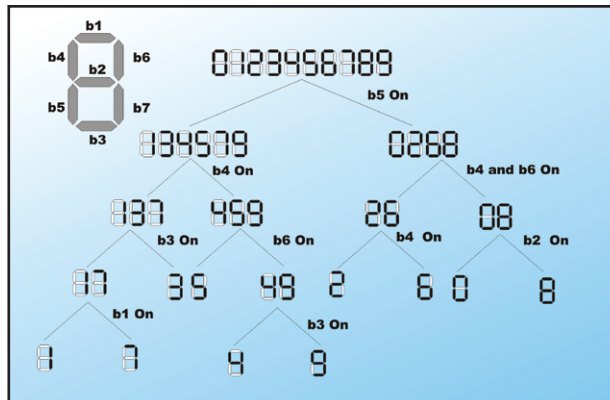


Figure 4: A decision tree to decide which digit is displayed

possible values so a series of tests leaves you with just one possible value. We'll use the test sequence illustrated by Figure 4 above, but there are other possible decision trees. Start at the top of the tree and go down each True or False branch until you achieve a single-digit result.

TREE'S A CROWD

Decision trees are relatively easy to convert into a set of nested If, Then, Else statements. See if you can understand how the following code reflects the structure of the decision tree:

```
If b5 Then#
  If b4 AND b6 Then#
    If b2 Then d=8 Else d=0#
  Else#
    If b4 Then d=6 Else d=2#
  End If#
Else#
  If b4 Then#
    If b6 Then#
      If b3 Then d=9 Else d=4#
    Else#
```

```
d=5#
End If#
Else#
  If b3 Then#
    d=3#
  Else#
    If b1 Then d=7 Else d=1#
  End If#
End If#
End If#
SampleDigit=d#
End Function#
```

At the end of each path through the tree, the corresponding value is stored in the variable d. SampleDigit is now

able to read a single digit from a meter. To make it work reliably, you may need to modify the threshold t in the SampleDigit routine. However, as long as the grid of crosswires is positioned reasonably accurately and the lighting is OK, it should work. It is surprisingly robust for such a simple technique.

Crosswire positioning with the camera is a lot easier if the camera is on a tripod. You can easily extend the program to read in multiple digits: just call SampleDigit with parameters set to place the crosswires on each of the digits you want to read in turn. It is easier to do this if the user interface is modified to allow the crosswires to be positioned and sized interactively. **CS**



Figure 3: The target crosswires correctly positioned on the digit to be read

CONTACT

MIKE JAMES

Mike has written several books on computing and programming and has been a senior lecturer at Teesside University

mike@computershopper.co.uk